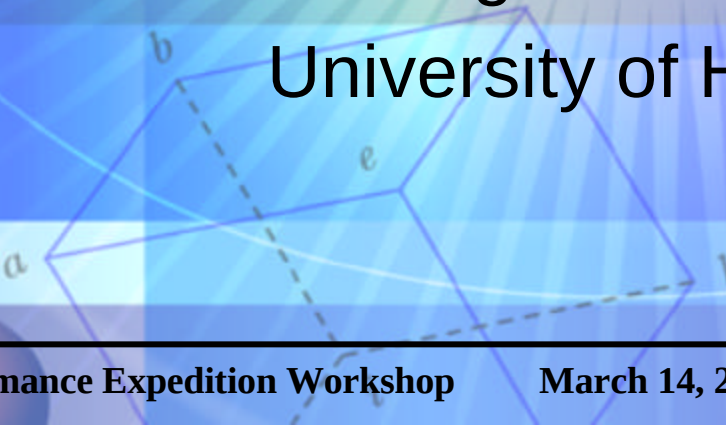
Application and Platform Adaptive Scientific Software

Lennart Johnsson
Dragan Mirkovic
University of Houston



Challenges

- Diversity of execution environments
  - Growing complexity of modern microprocessors.
    - Deep memory hierarchies
    - Out-of-order execution
    - Instruction level parallelism
  - Growing diversity of platform characteristics
    - SMPs
    - Clusters (employing a range of interconnect technologies)
    - Grids (heterogeneity, wide range of characteristics)
- Wide range of application needs
  - Dimensionality and sizes
  - Data structures and data types

Challenges

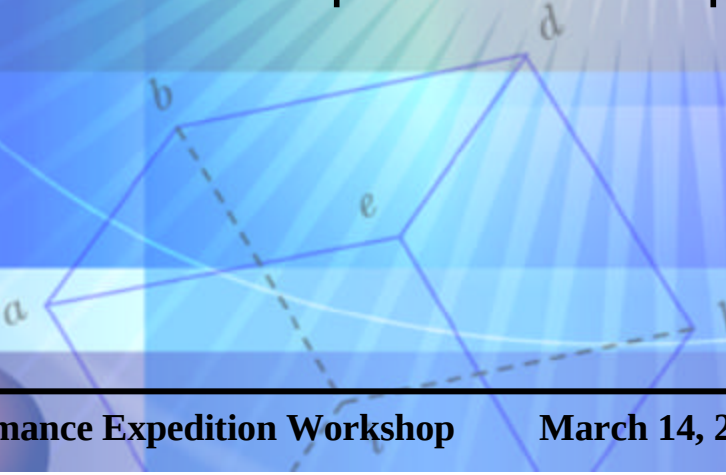
- Algorithmic
  - Unfavorable data access pattern (big 2^n strides)
  - High efficiency of the algorithm
    - low floating-point v.s. load/store ratio
  - Additions/multiplications unbalance
- Version explosion
  - Verification
  - Maintenance

Approach

- Automatic algorithm selection – polyalgorithmic functions
- Code generation from high-level descriptions
- Extensive application independent compile-time analysis
- Integrated performance modeling and analysis
- Run-time application and execution environment dependent composition
- Automated installation process

Approach

- Code preparation at installation (platform dependent)
- Integrated performance models and data bases
- Algorithm selection at run-time from set defined at installation
- Program construction at run-time based on application and performance predictions



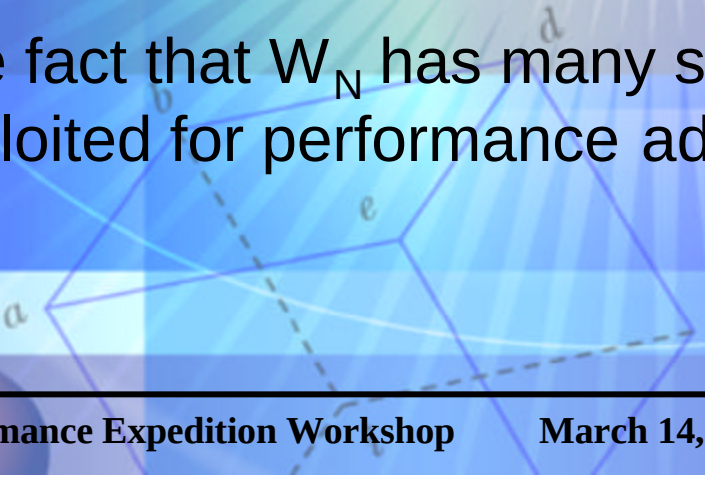
The UHFFT
An Adaptive FFT Library

- Application of $\mathbf{W_N}$ requires $O(N^2)$ operations
- Fast algorithms use sparse factorizations of W_N,

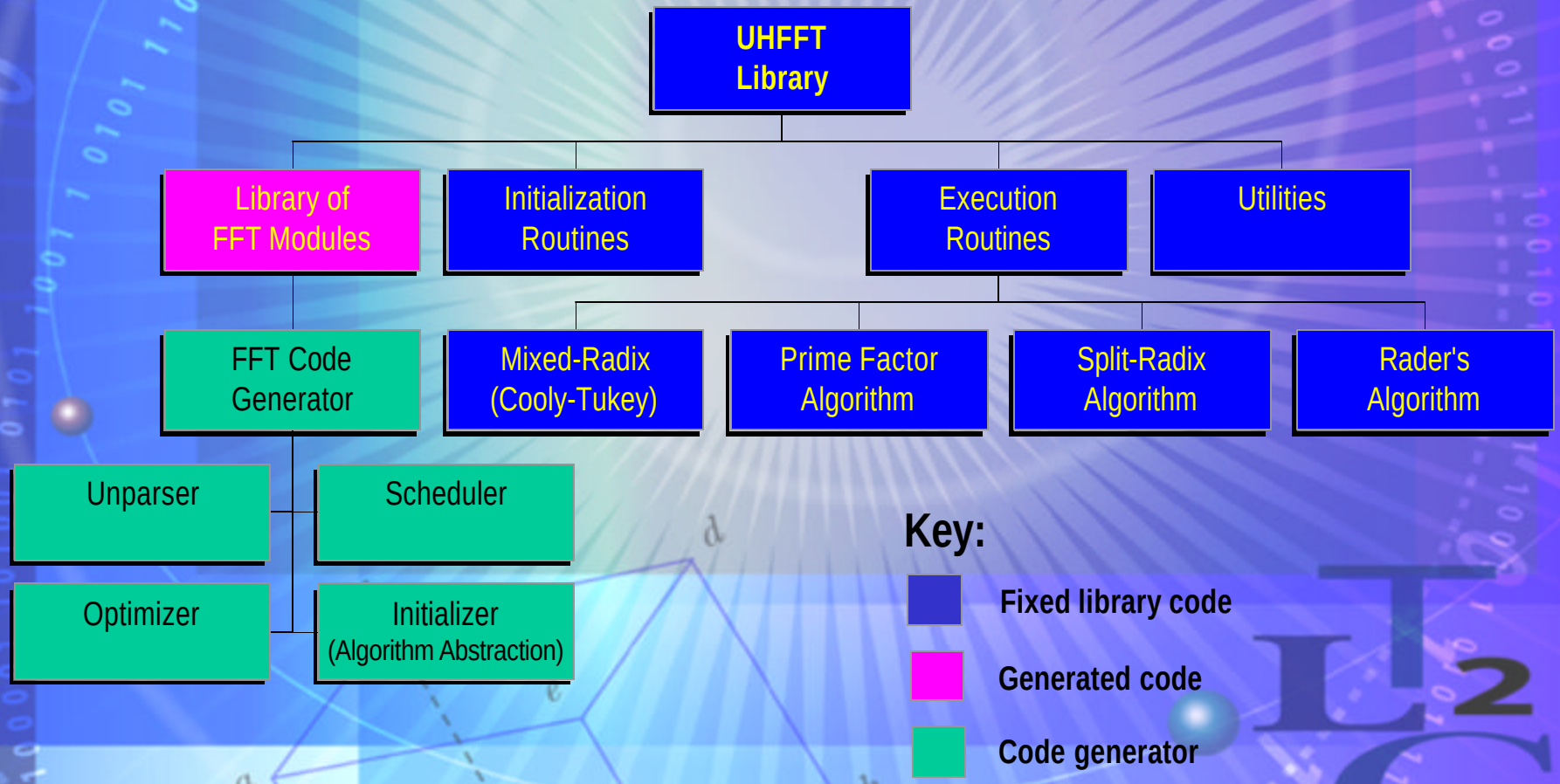
  $W_n = A_1 A_2 \ldots\ldots A_k$, where A_i's are sparse

  and requires O(n) operations and k=O(logN)

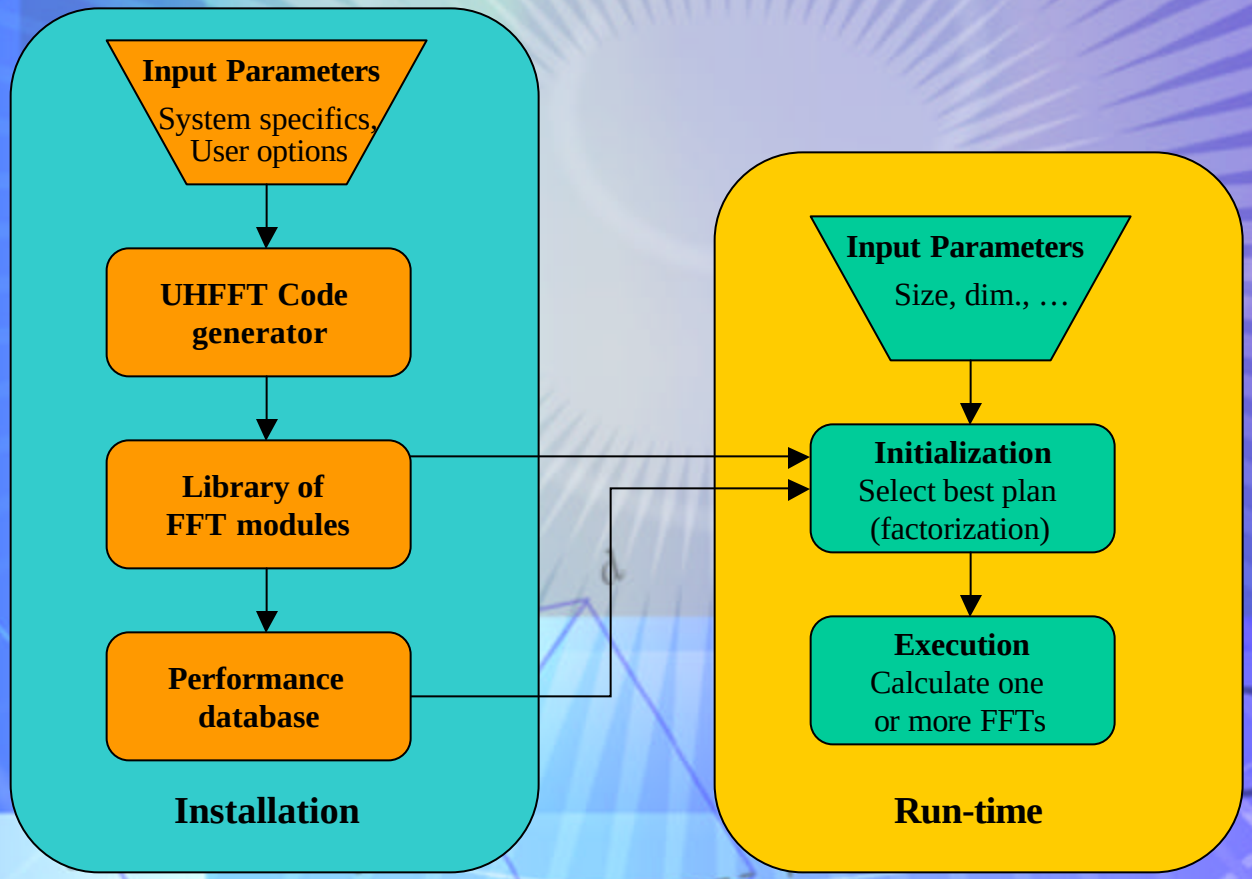
- The fact that W_N has many sparse factorizations is exploited for performance adaptivity



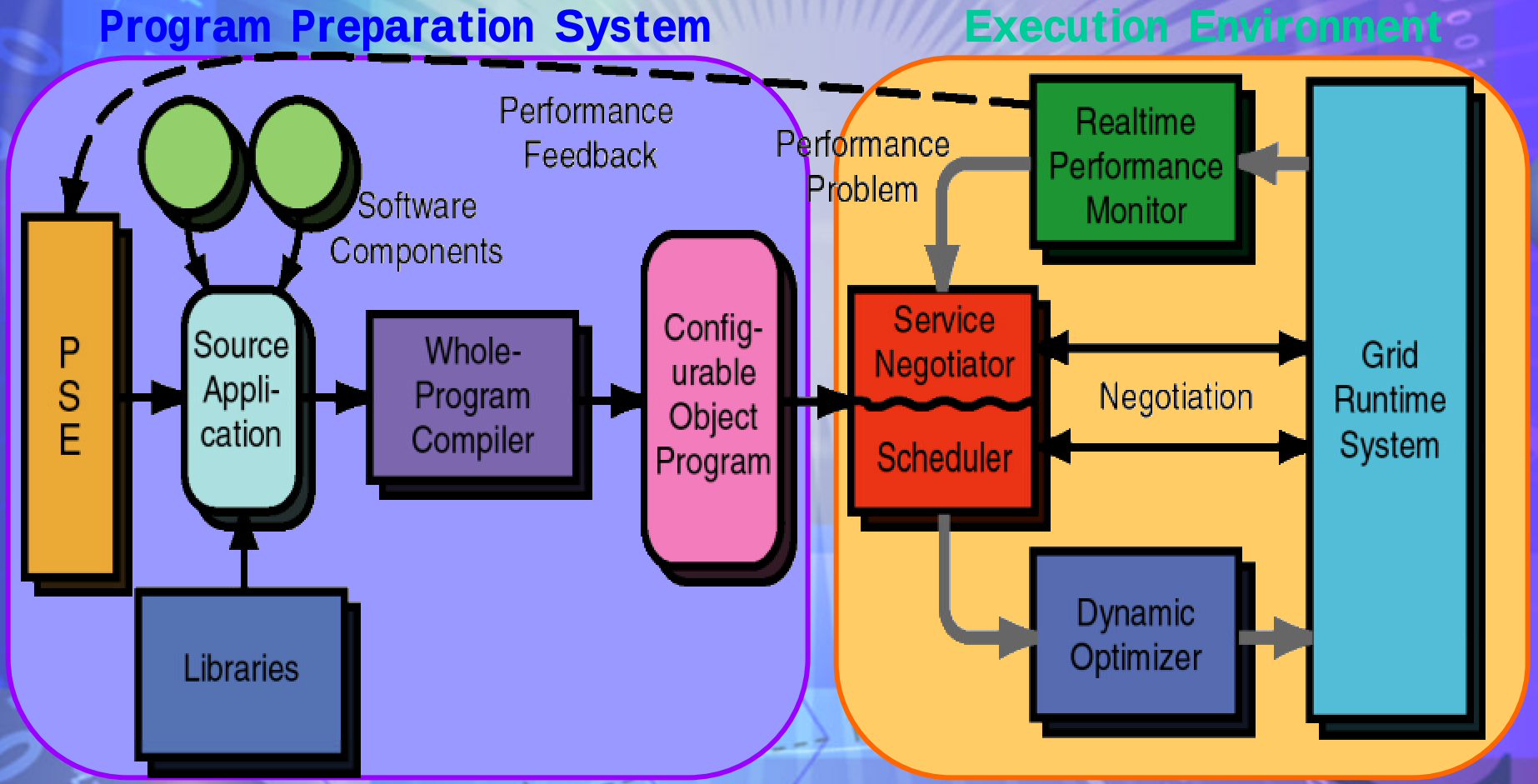
UHFFT Library Architecture



Performance Tuning Methodology



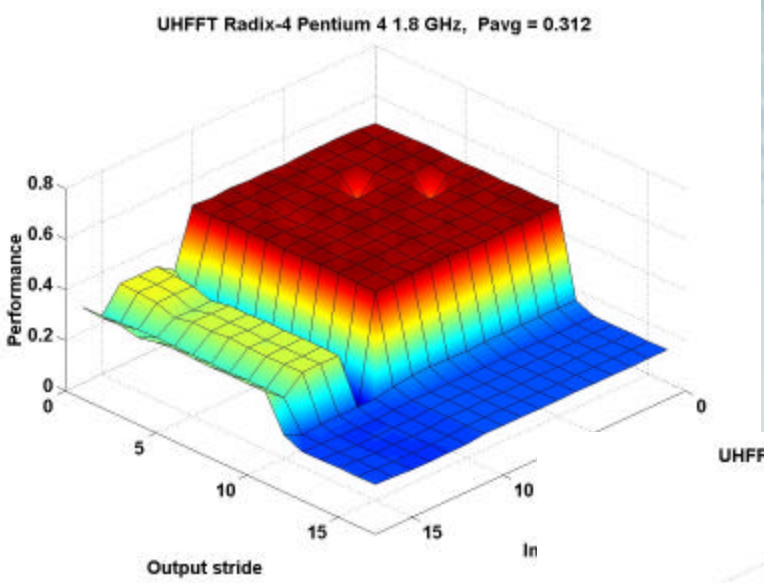
Grid Application Development Software (GrADS)



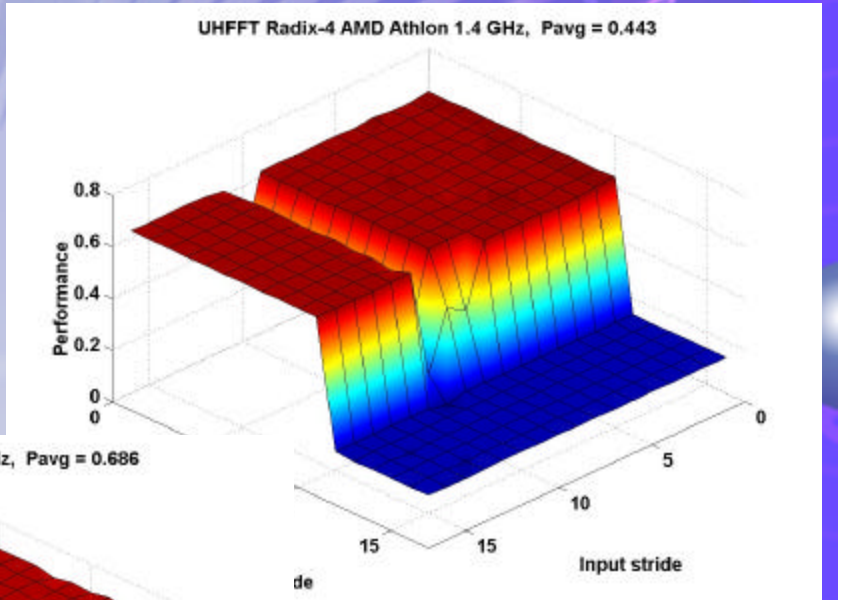
Characteristics of Some Target Architectures

Processor	Clock frequency	Peak Performance	Cache structure
Intel Pentium IV	1.8 GHz	1.8 GFlops	L1: 8K+8K, L2: 256K
AMD Athlon	1.4 GHz	1.4 GFlops	L1: 64K+64K, L2: 256K
PowerPC G4	867 MHz	867 MFlops	L1: 32K+32K L2: 256K, L3: 1-2M
Intel Itanium	800 Mhz	3.2 GFlops	L1: 16K+16K L2: 92K, L3: 2-4M
IBM Power3/4	375 MHz	1.5 GFlops	L1: 64K+32K, L2: 1-16M
HP PA 8x00	750 MHz	3 GFlops	L1: 1.5M + 0.75M
Alpha EV67/68	833 MHz	1.66 GFlops	L1: 64K+64K, L2: 4M
MIPS R1x000	500 MHz	1 GFlop	L1: 32K+32K, L2: 4M

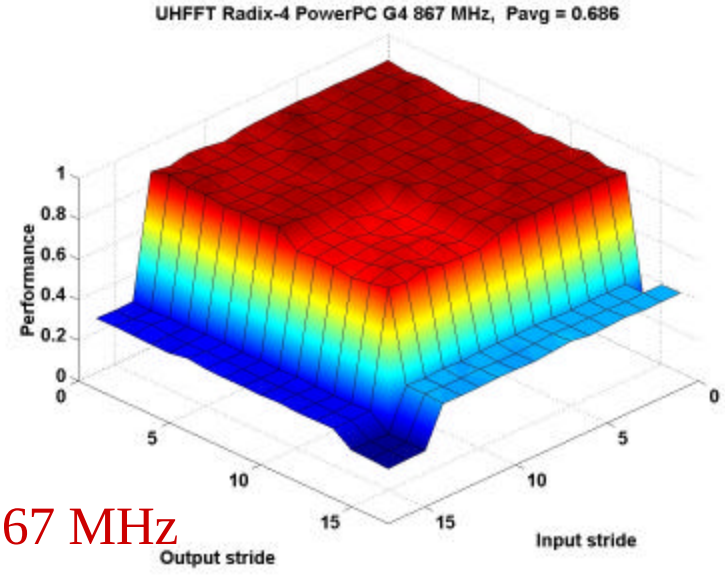
Radix-4 codelet performance, 32-bit architectures



Intel PIV 1.8 GHz

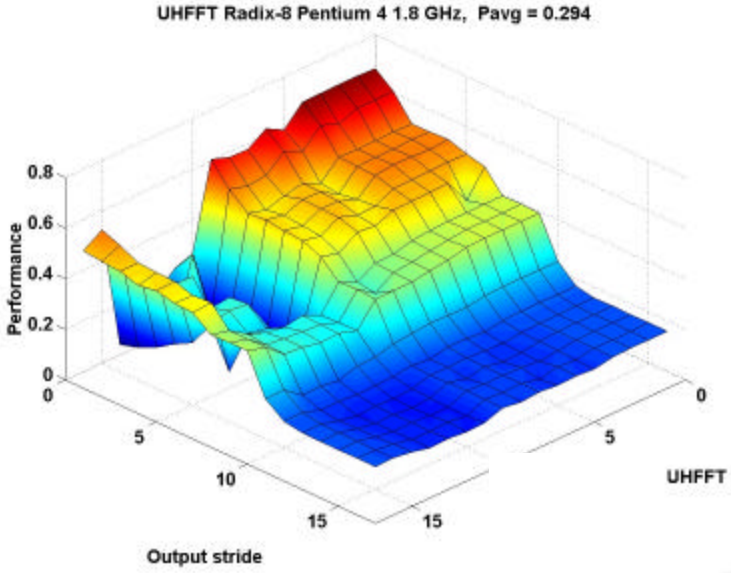


AMD Athlon 1.4 GHz

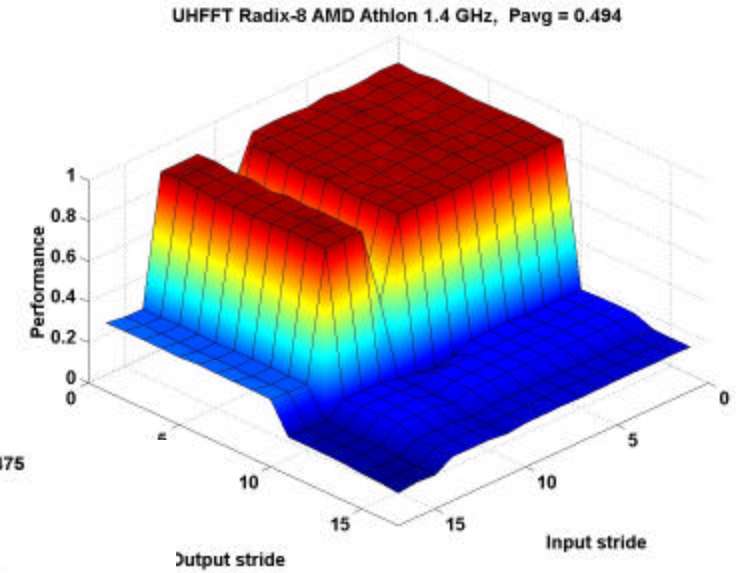


PowerPC G4 867 MHz

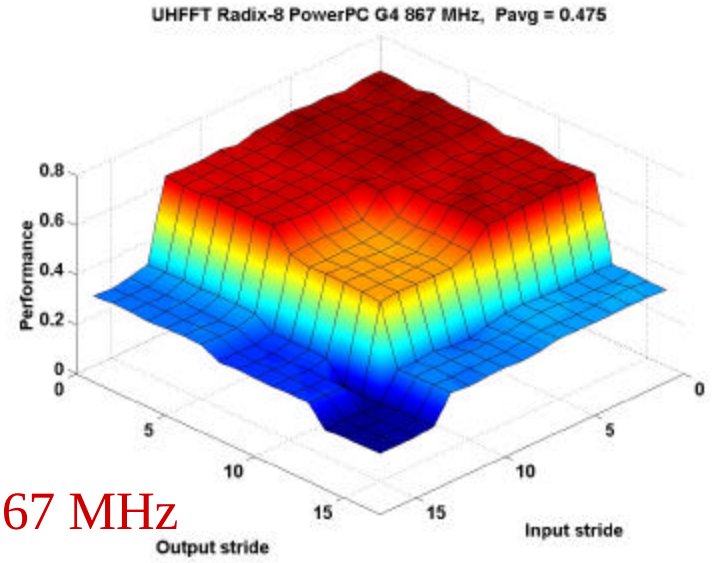
Radix-8 codelet performance, 32-bit architectures



Intel PIV 1.8 GHz

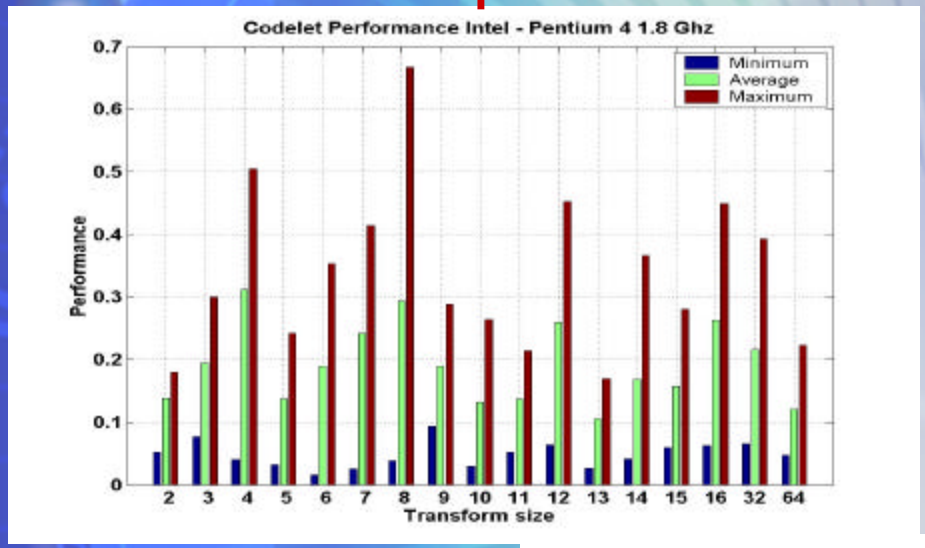


AMD Athlon 1.4 GHz

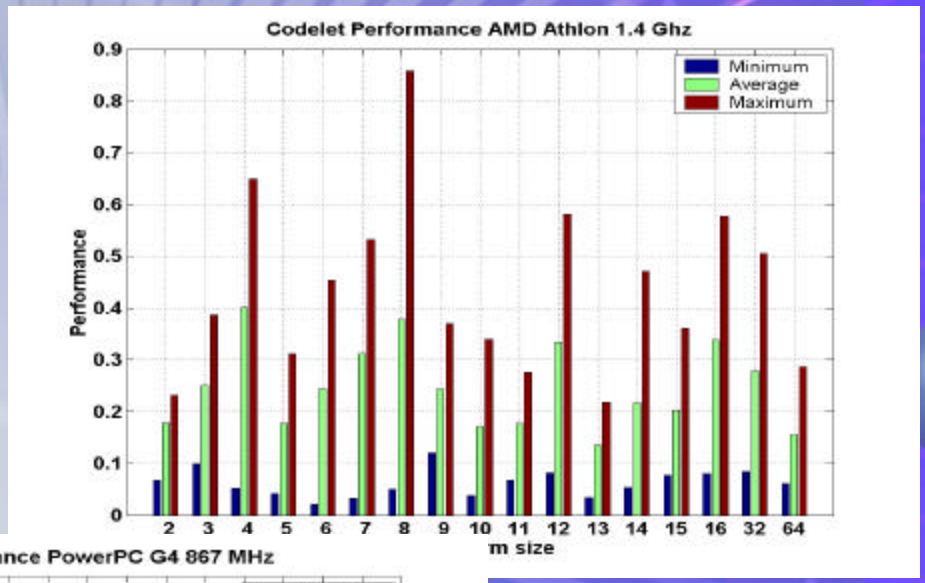


PowerPC G4 867 MHz

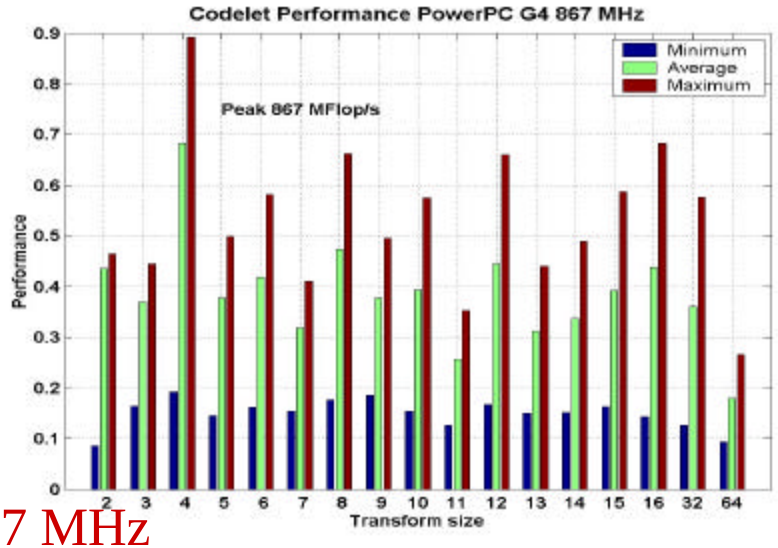
Codelet performance 32-bit architectures



Intel PIV 1.8 GHz

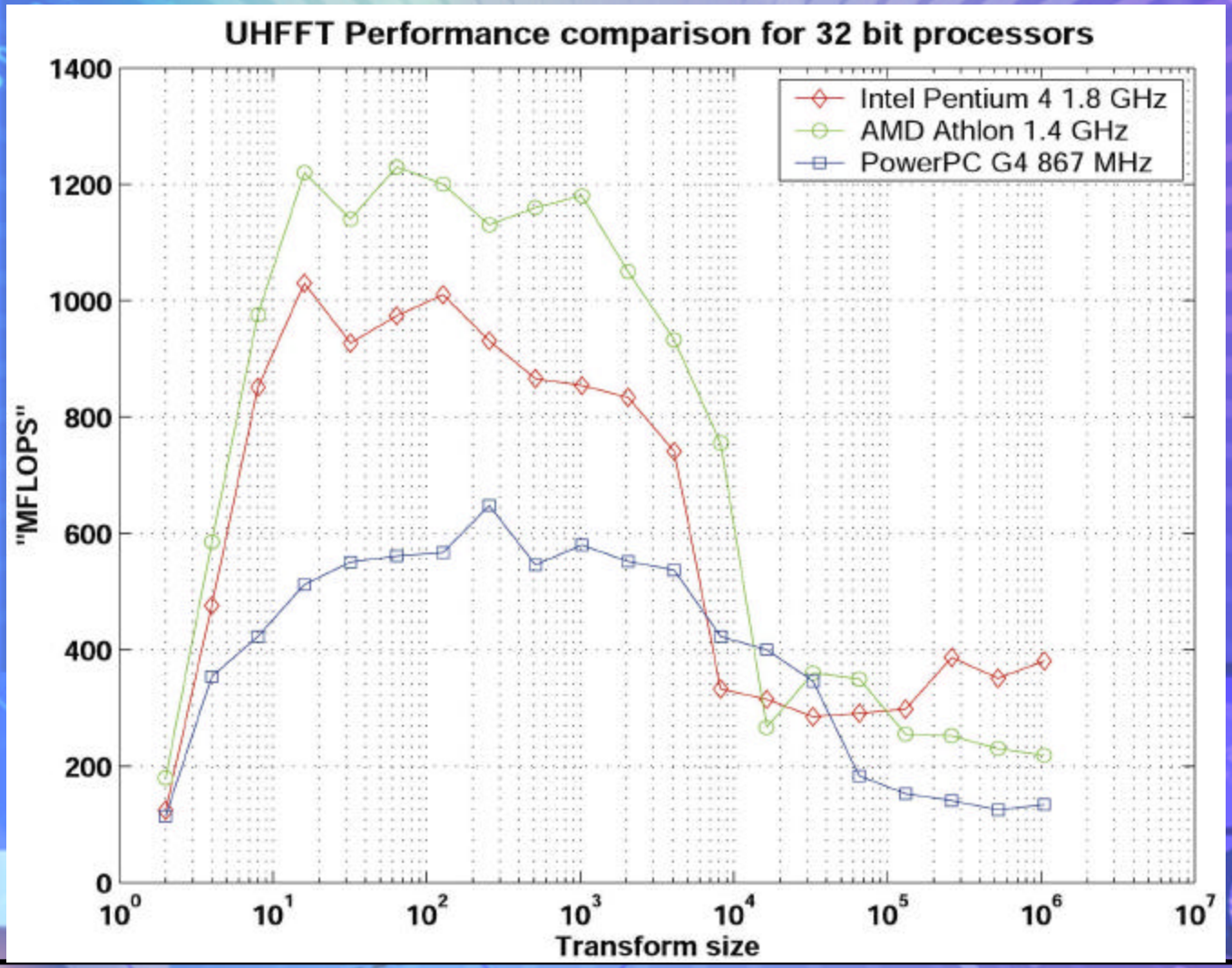


AMD Athlon 1.4 GHz



PowerPC G4 867 MHz

Plan Performance, 32-bit Architectures

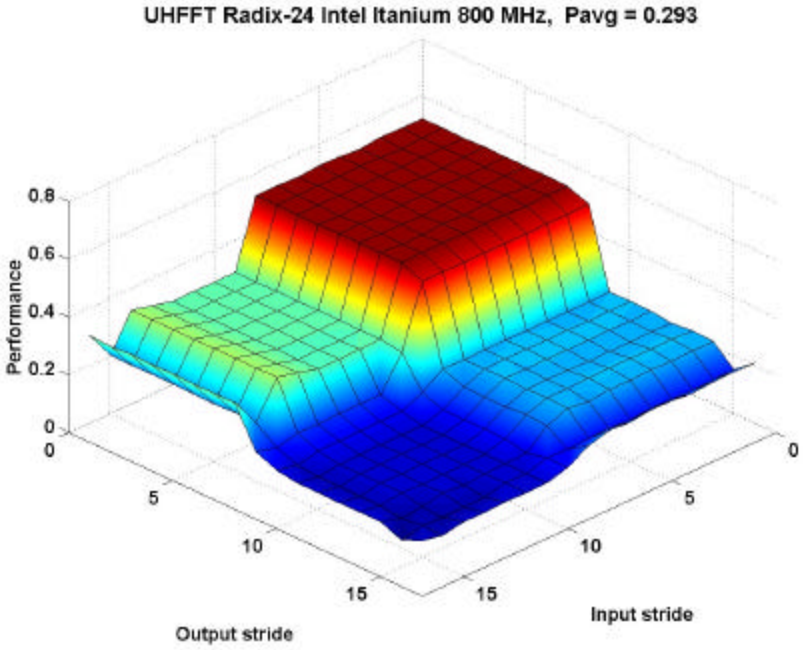


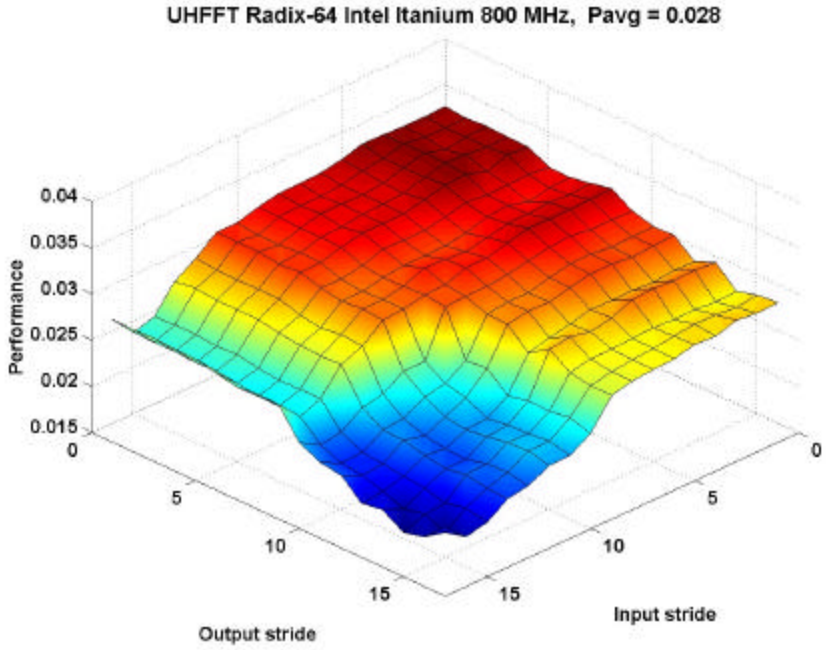
Itanium

- Intel Itanium 800 MHz
  - 2 GB SDRAM
  - 2 MB of L3 cache
  - Bus speed: 133 MHz
  - Inherent parallelism in IA-64
  - Multiple FPUs with fused multiply-add instructions
  - Large number of registers provide good support for ILP
  - Relatively small L1 cache (16k+16k)
    - Large codelets do not perform very well
  - Complex scheduling problem
    - Cache reuse and parallelism have opposite requirements
  - OS: HP-Unix 11i version 1.5
  - Compiler: gcc version 2.96
  - Compiler options: -O2 –fomit-frame-pointer –funroll-all-loops

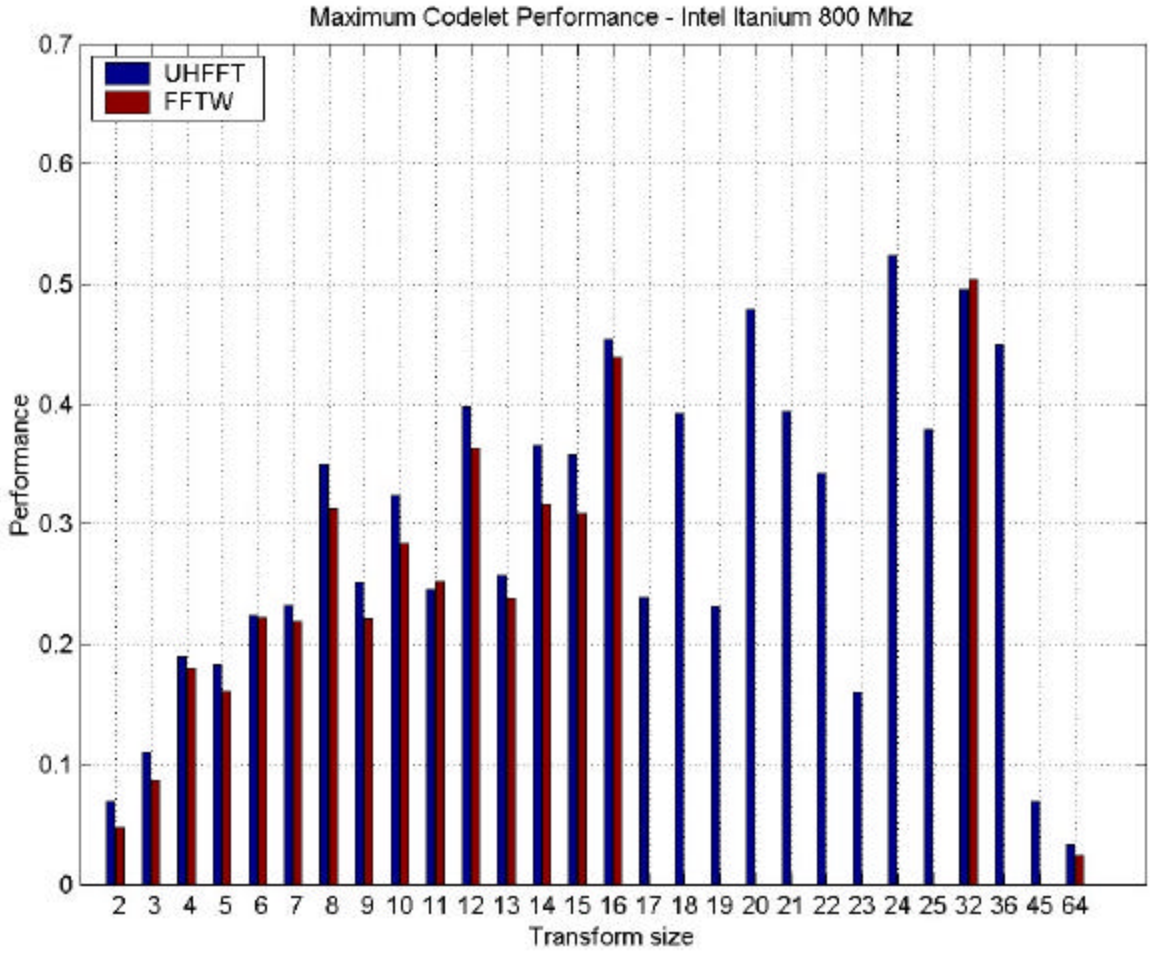
Itanium Codelet performance examples

Best and "worst"

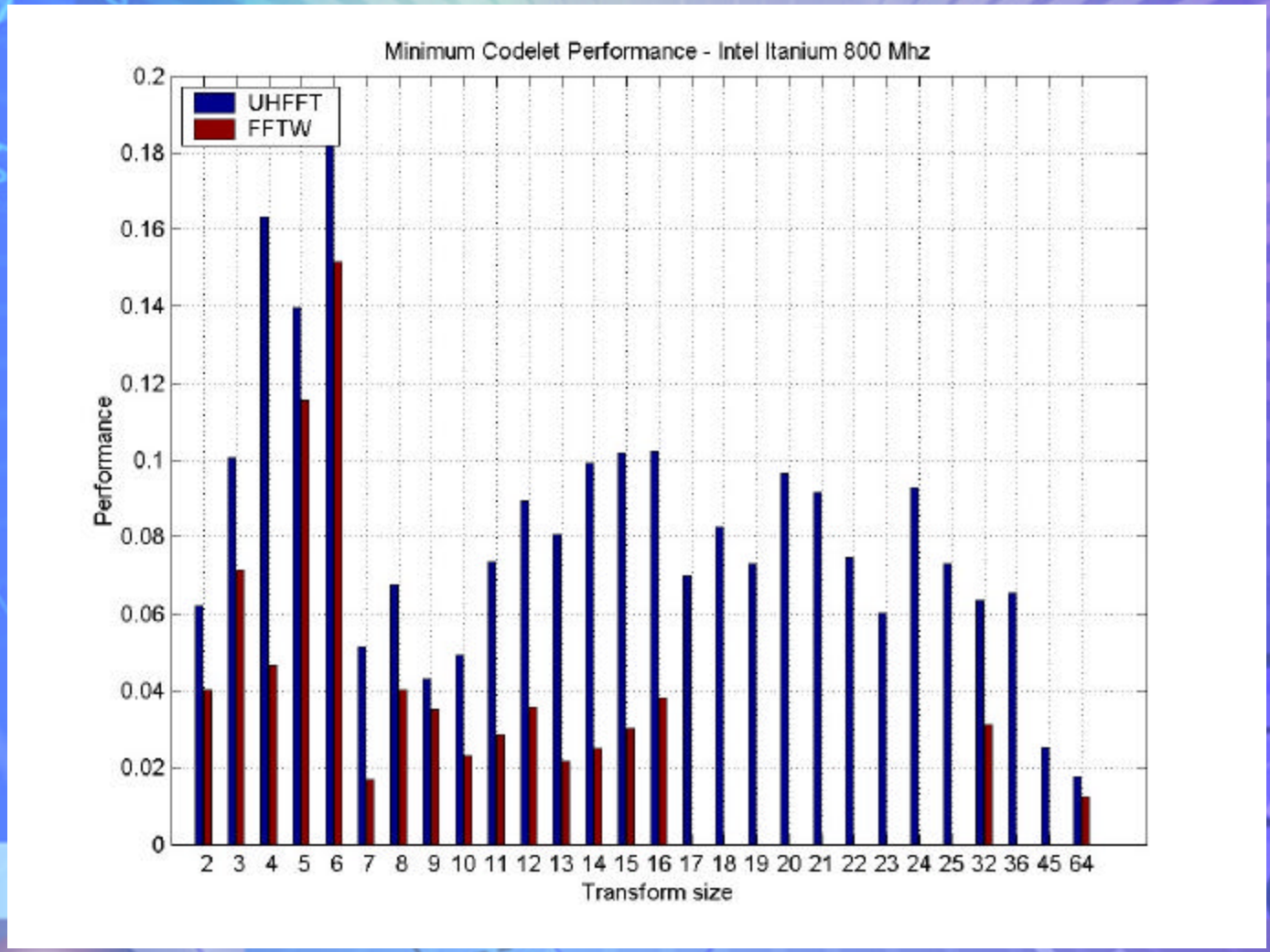




Itanium maximum codelet performance



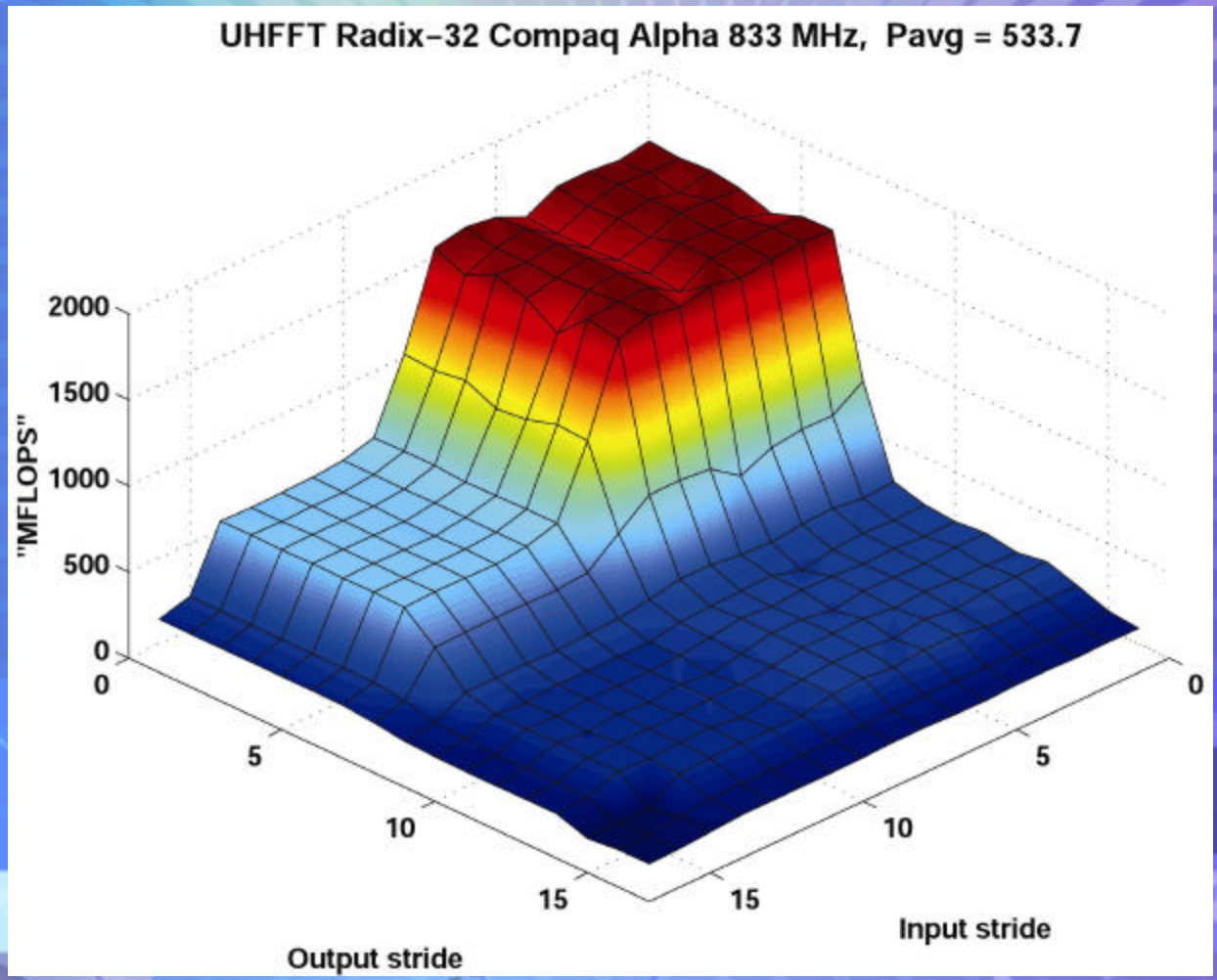
Itanium minimum codelet performance



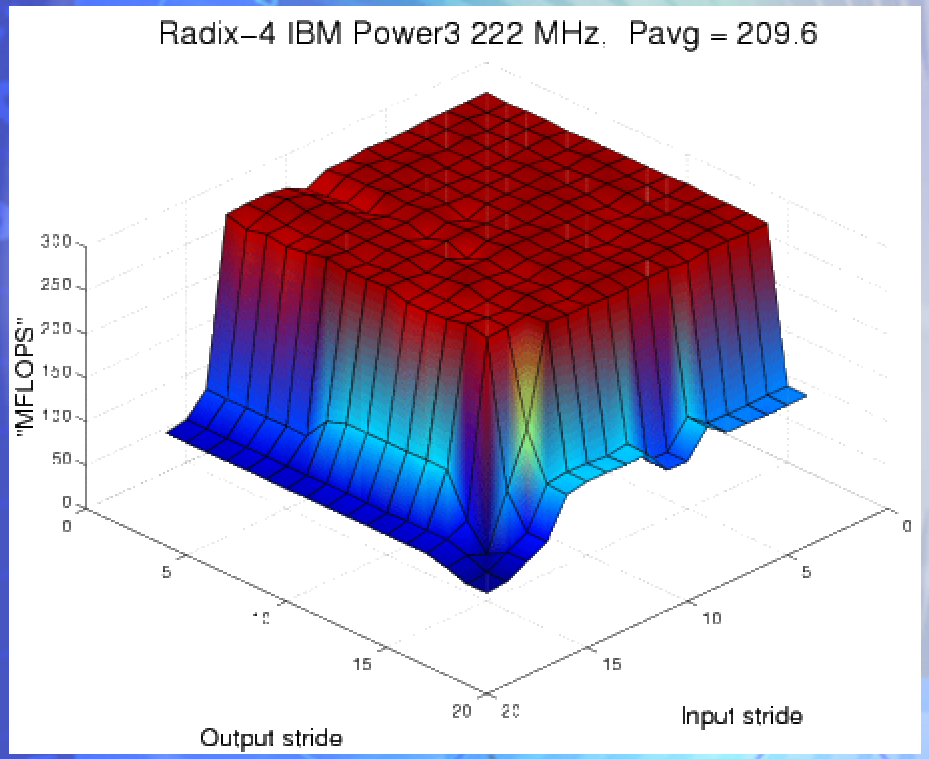
Alpha

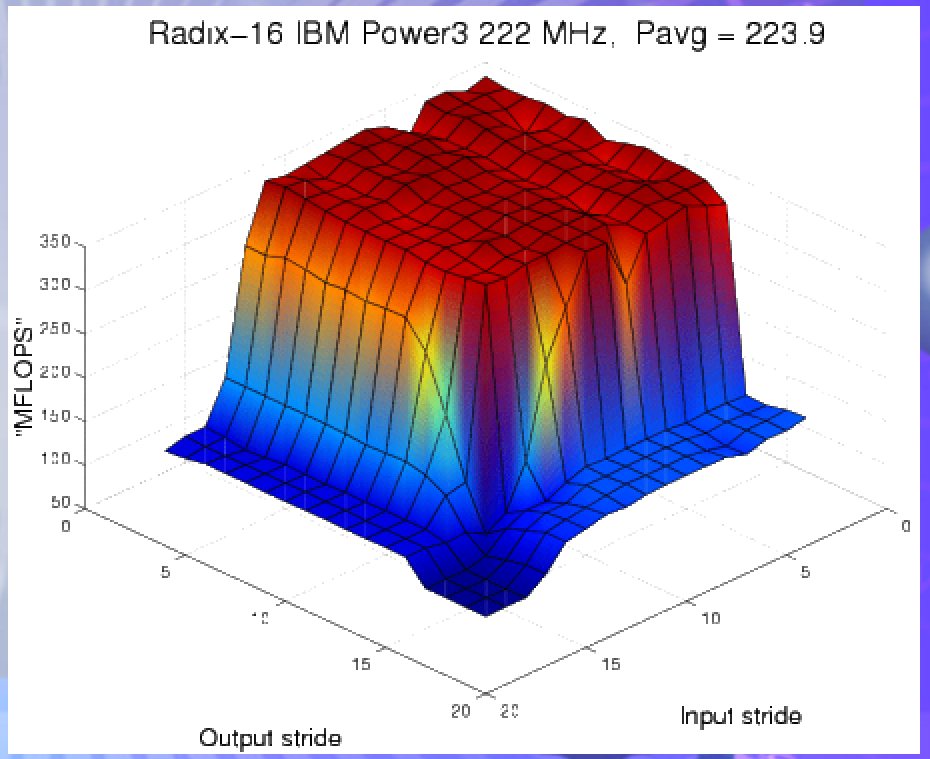
- Compaq Alpha 833 MHz
  - 2 Gb SDRAM
  - Bus speed: 133 MHz
  - OS: True64 Unix
  - Compiler: gcc version 2.96
  - Compiler options: -O2 –fomit-frame-pointer –funroll-all-loops
  - Complex–to-complex, out-of-place, double precision transforms
  - Codelet sizes: 2 – 25, 32, 36, 45, 64
  - Strides: $2^{[0-16]}$
  - Performance:
    - Absolute: $5*n*\log(n)/t_{CPU}$ in “FLOPS”
    - Relative: Absolute/(Peak performance of the processor)
  - Peak performance: 1.66 GFLOPS

Alpha codelet performance example



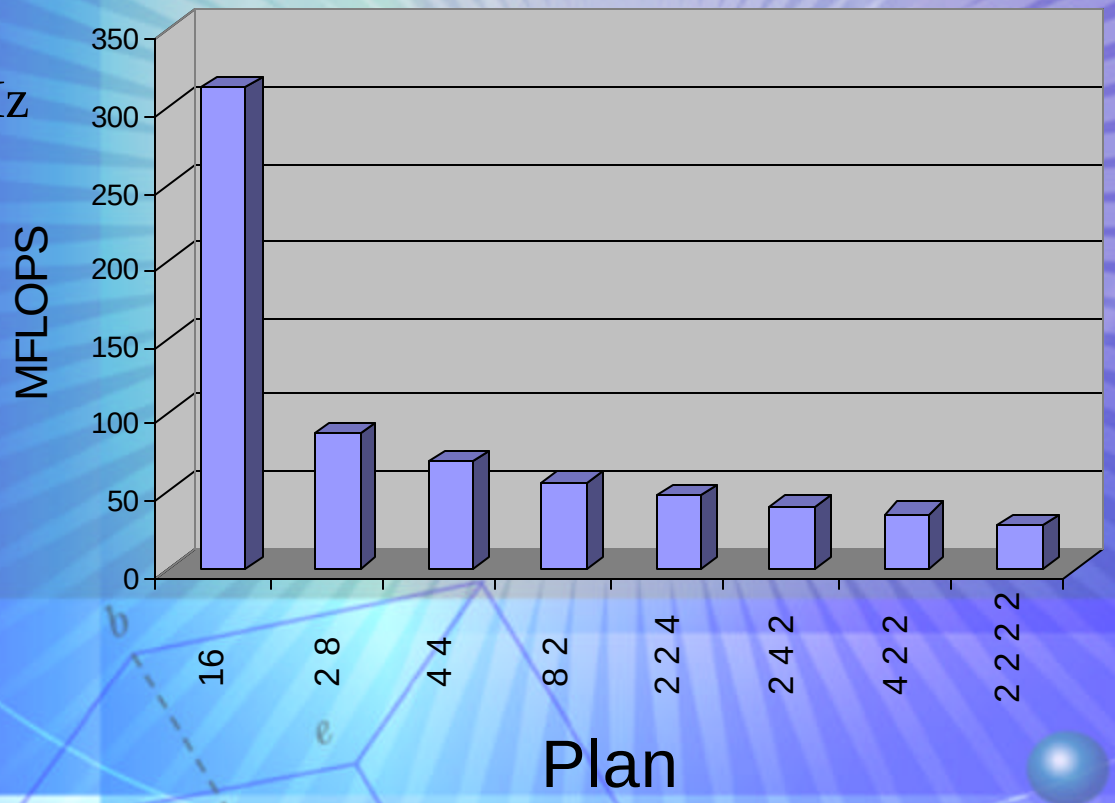
Power3 codelet performance examples





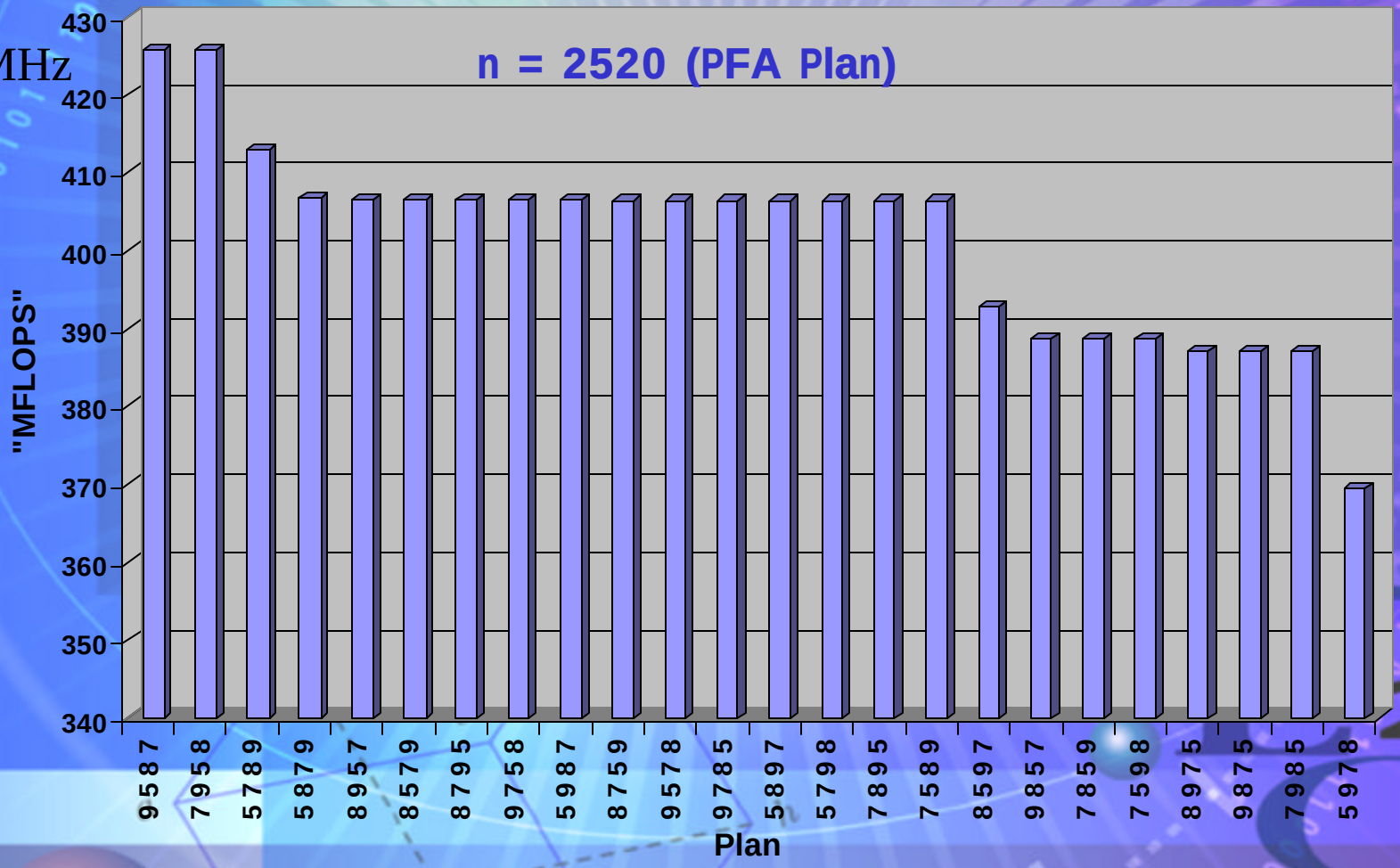
Power3 plan performance example

222 MHz

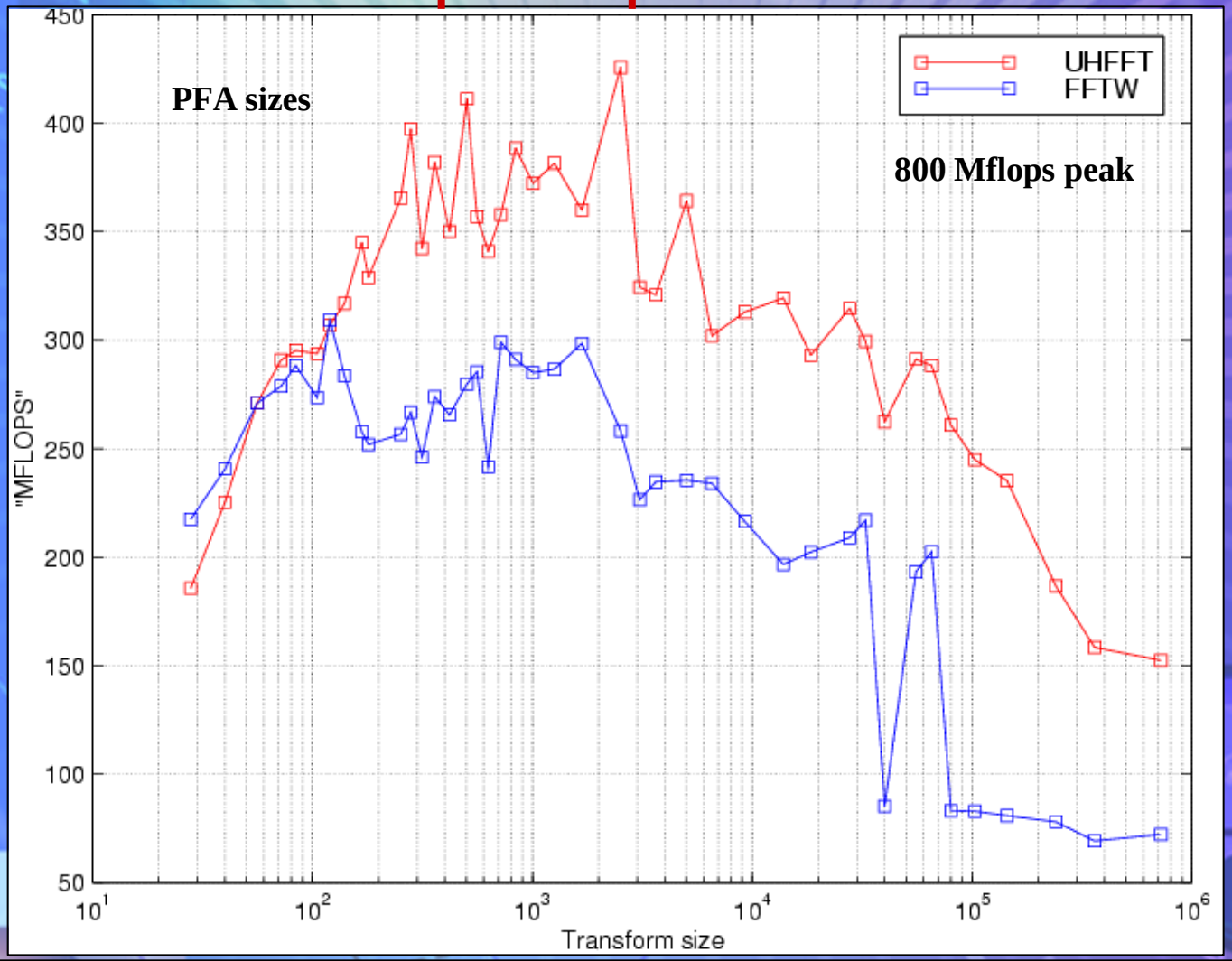


Power3 plan performance

222 MHz



Power3 plan performance



Advantages of the UHFFT Approach

- Code generator written in C
- Code is generated at installation
- Codelet library is tuned to the underlying architecture
- The whole library can be easily customized through parameter specification
  - No need for laborious manual changes in the source
  - Existing code generation infrastructure allows easy library extensions
- Future:
  - Inclusion of vector/streaming instruction set extension for various architectures
  - Implementation of new scheduling/optimization algorithms
  - New codelet types and better execution routines
  - Unified algorithm specification on all levels

Advantages of the UHFFT Approach

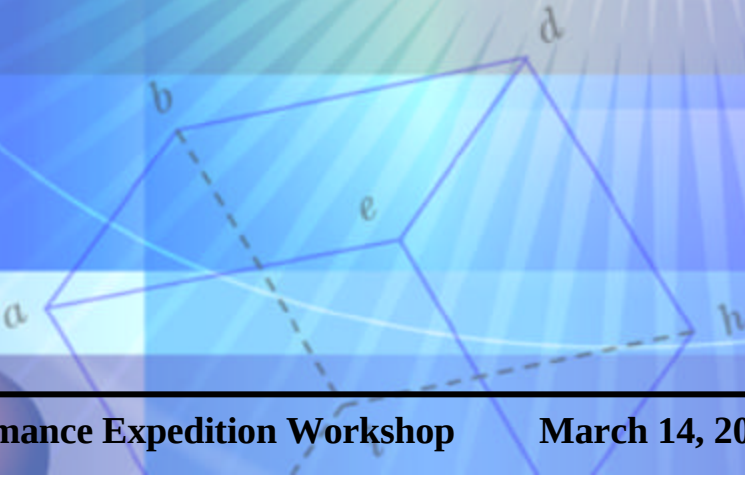
- UHFFT employs more ways of combining codelets for execution than any other library
- Better coverage of the space of possible algorithms
- The PFA algorithm yields good performance where the Mixed-Radix algorithm (MR) performs poorly
  - PFA algorithm requires less FP operations than MR
  - Data access pattern in PFA is more complex than in MR, but large 2^n strides can be avoided
- Example IBM Power3
  - Good: 128-way set associative L1 data and instruction caches
  - Bad: Direct mapped L2 cache very vulnerable to cache trashing despite the large cache size
  - PFA execution model works better for large FFT sizes

Contemplated Extensions

- Extension to sine and cosine transforms
- Further optimization of parallel aspects
- Extension to Multigrid methods
- Extension to Wavelets
- Extension to Convolution
- Extension to Lagrangian Finite Elements
- Extensions to Spherical Transforms
- Toolbox for code generation and optimization for FFT, …
- Extension to parallel programming paradigms other than MPI

Related Efforts

- Wassem
- CMSSL
- CWP
- FFTW
- Spiral



The UHFFT: An Adaptive FFT Library

- UHFFT Web site:
  - http://www.cs.uh.edu/~mirkovic/uhfft
- Publications

[1] Mirkovic, D., Johnsson S.L. (2001) Automatic Performance Tuning in UHFFT Library. In proceedings of the 2001 International Conference on Computational Science, ICCS 2001, May 2001, San Francisco, USA, Lecture Notes in Computer Science 2073, Vol. 1, pp. 71-80.

[2] Mirkovic, D., Mahasoom R., Johnsson S.L. (2000) An Adaptive Software Library for Fast Fourier Transforms. Proceedings of the 2000 International Conference on Supercomputing, Santa Fe, NM , pp. 215-224.